Min Cheol Kim
Joseph Wu

Predicting and Understanding User Behavior in Twitch TV Sessions

Dataset Summary

In this project, we want to investigate audience behavior in eSports by analyzing data from Twitch, one of the biggest live streaming platform dedicated to videogames. We are using the dataset “Twitch Sessions”, a parsed dataset taken from Twitch API on a public data repository hosted by Telecom Bretagne, a French research center specializing in Information Technology and telecommunication. Since the data were originally taken from Twitch and minimally processed by a reliable institution, we are fairly confident about the fidelity and completeness of the data.

“Twitch Sessions” contains a series of time-stamped data, taken every second over a day, of every streaming session occurring at that time. Each entry has 28 attributes summarizing the streaming session, including channel information, viewers, video bitrate, category, etc.

We chose a week’s worth of data from January 6, 2014 (Monday), to January 12, 2014 (Sunday). That is 7 days of data, approximately 1 million rows per day. We plan to make observations about audience behavior over the cycle of a day as well as of a week.

Our reserved test set will be same data over the next week of days, from January 13th (Monday), 2014 to January 19th (Sunday).

Variables (Continuous vs. Binary)

One variable we plan on using as a continuous response variable is the number of viewers at each streaming session. This variable ranges from the low teens to upwards of hundred thousands who tune in for different channels and sessions. We want to use these exact numbers possibly for regression modelling and prediction, so it’s best to keep it as a continuous variable.

One variables we plan on using as a binary response variable is whether a streaming session is flagged as ‘mature’. We will be using this binary variable to explore how the audience behavior and composition might change due to the nature of the content, characterized by whether the content is intended for mature audience or not.

Other continuous variables from the dataset include upload time, video bitrate, total

channel views. Another binary variable to use is whether the channel is featured on Twitch. Other variables, such as geographical location, content category, will be used as categorical variables.

How we plan to use data

We plan to execute some descriptive analysis as well as predictive analysis on the “Twitch Sessions” dataset. These analyses will help us uncover what makes the big stars in eSports, what type of content the audience is drawn to, and how different factors contribute to the success of certain video game streaming channels. Below are some preliminary directions we plan to explore with the dataset.

Descriptive Analysis:

- How does total audience fluctuate throughout a day? And throughout a week?

- How do sessions differ with respect to whether they are for mature audience? Whether they are featured on Twitch?

- What do the most successful channels do differently from channels that underperform?

- How do videos of different categories perform on Twitch?

Predictive Analysis:

- Can we predict # of viewers who tune in on a channel based on attributes besides their channel name (upload time, category, feature, etc.)? What type of regression modeling is best suited for this prediction?

- Are there ways to cluster the sessions in an intuitive way?

Preliminary Data Exploration Results

- The only N/A or none values are found in the “geo” column, representing the country of the streaming. We will not use the rows that contain the “none” values when we are using the geo column.

- There are several columns that are seemingly irrelevant, such as the ones generated by the Twitch system and are arbitrary. One example is the channel id. The channel name provides a categorical variable we can work with, but the id is simply an arbitrary and unique number assigned to each channel. The audio and the video codec also seem to be irrelevant columns.

- A cursory analysis from covariate pairs plots and our intuition led to the analysis of the following covariate associates: Mature vs Featured, Mature vs Viewers, Channel views vs Viewers and Featured vs. Viewers.

- We found that the sessions that were flagged as “mature” on average had a greater number of maximum viewers (the peak number of viewers on a given session). The mean for “mature” flagged sessions was 25 viewers while that of the non-mature sessions was about 14 viewers.

- There seemed to be an association between the mature and the featured flag. Out of the 185878 that were featured, around 28% or 51780 sessions were flagged as mature.

- As expected, the sessions that were featured had, on average, several times the amount of maximum viewer traffic compared to the non-featured sessions. This is also interesting since the “mature” and the “featured” flags both are associated with increased view count, but they themselves are inversely associated.

- As we continue to explore and analyze the data, several interaction variables could be added. A combination of “mature” and “featured” flags could be added; for example, we could have a covariate that indicates whether the session is both “mature” and “featured.” One other interaction term we can foresee now is with video bitrate and video uptime, since they both concern the quality of the streaming experience.

- One population model we can suggest for predicting the number of viewers is a linear combination of all of our factors and all of their interaction terms/variables. This is certainly dependent on the fact that we are first attempting a prediction problem. For example, we would not intuitively believe that having a high channel view would somehow make the content more interesting and therefore attract more viewers; however, someone who makes interesting sessions probably always have a high number of session viewers as well as a large number of channel viewers.

- The code used for the preliminary analysis above can be found on the following pages.

Code / Output

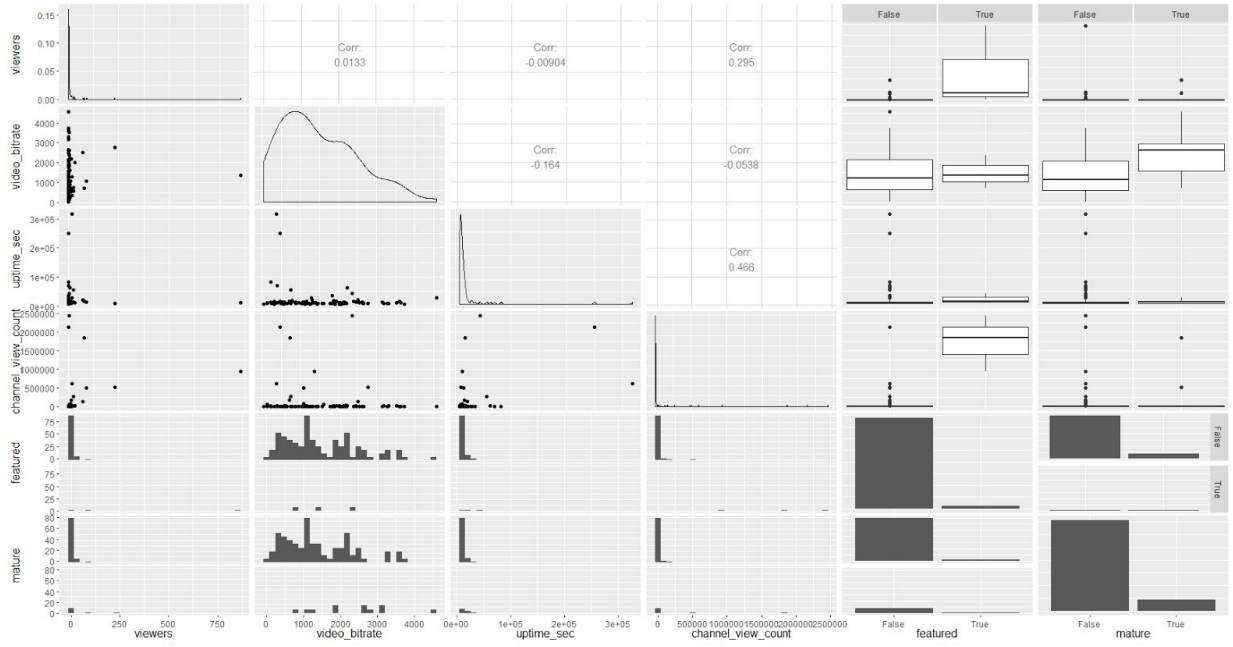
```
library(tidyverse)
library(GGally)

# Read and clean data ------------------------------------------------------------

load("C:/Users/minch_000/Desktop/226 project/.RData")

stream_data <- select(week_data, -accessed_at_utc, -channel_login, -video_height,
                      -video_width, -embed_count, -site_count, -audio_codec,
                      -video_codec, -session_count)
levels(stream_data$mature) <- c("False", "True")
stream_data$viewers <- as.numeric(as.character(stream_data$viewers))

# Finding associated covariates
assoc_data <- select(stream_data, viewers, video_bitrate, uptime_sec, channel_view_count,
featured, mature)
sample_assoc_data <- assoc_data[sample(nrow(assoc_data), 100),]
ggpairs(sample_assoc_data)
```



```
# Group by session for future uses
by_session <- group_by(stream_data, session_id)

# Find out how having a "Mature" flag correlates with max # of viewers
# of a given session.
by_mature <- group_by(stream_data, session_id) %>%
             summarise(
               max_viewer = max(viewers),
               mature = mature[1]
             ) %>%
             filter(
               !is.na(max_viewer),
               !is.na(mature)
             ) %>%
             group_by(mature) %>%
             summarise(
               max_viewer_avg = mean(max_viewer)
             )
by_mature
```

```
# A tibble: 2 × 2
  mature max_viewer_avg
  <fctr>         <dbl>
1  False      13.87001
2   True      25.28092
```

```
# Association between the mature and the featured flag
by_feature_mature <-  select(by_session,
                       featured,
                       mature
                     ) %>%
                     filter(featured == 'True') %>% # only consider featured sessions
                     group_by(mature) %>%
                     summarise(
                       count = n()
                     )
by_feature_mature
```

```
# A tibble: 2 × 2
  mature  count
  <fctr>  <int>
1  False 134098
2   True  51780
```

```
# Find out how having a "Featured" flag correlates with max # of viewers
# of a given session.
by_featured <- group_by(stream_data, session_id, featured) %>%
  summarise(
    max_viewer = max(viewers)
  ) %>%
  filter(
    !is.na(max_viewer),
    !is.na(featured),
    featured == 'False' | featured == 'True'
  ) %>%
  group_by(featured) %>%
  summarise(
    max_viewer_avg = mean(max_viewer)
  )
by_featured
```

```
# A tibble: 2 × 2
  featured max_viewer_avg
    <fctr>          <dbl>
1    False       8.734332
2     True     538.790890
```

```
# There were a total of 134098 + 51780 = 185878 sessions that were featured.
# Out of these, 51780, or 28%
```

Regression Task

Baseline

For the baseline regression model, we chose to always predict the sample mean, without any variable selection or transformation. We chose this method because we want to see what a result with minimal manipulation of the data would look like. Since some of the covariates are categorical and need to be transformed or dropped, we didn't want to meddle with the data too much for our baseline predictions.

Sample mean (peak viewer) = 15.18

RMSE error = 263.79

Variable Selection and Transformation

From the total of 16 variables, we first used our intuition as well as understanding of the dataset and the real work to weed out irrelevant covariates, such as 'session id' and 'channel id', which are just digits for unique identification purposes. We decided to drop 'timezone', because there are too many missing entries. We also converted 'geo' into a binary variable of whether the session took place in the U.S., since there are more than 146 geographical categories and U.S. composes more than a third of the entries, whereas some geographical categories had very few data points. We then observe the correlation matrix of the remaining variables and remove the variables that have high dependency (such as 'language_channel' and 'language_session'). In the end, 11 variables are used as covariates for regression modeling.

See Fig 1 for the GGPairs plot and Fig 2 for a covariate-response scatterplot. All of the continuous variables are heavy-tailed (outliers in the extremities) and would benefit from log transformation. We took the log transformation of the response variable ('max_viewer'), and all continuous covariates ('video_bitrate', 'uptime', 'uptime_sec', 'channel_view_count').

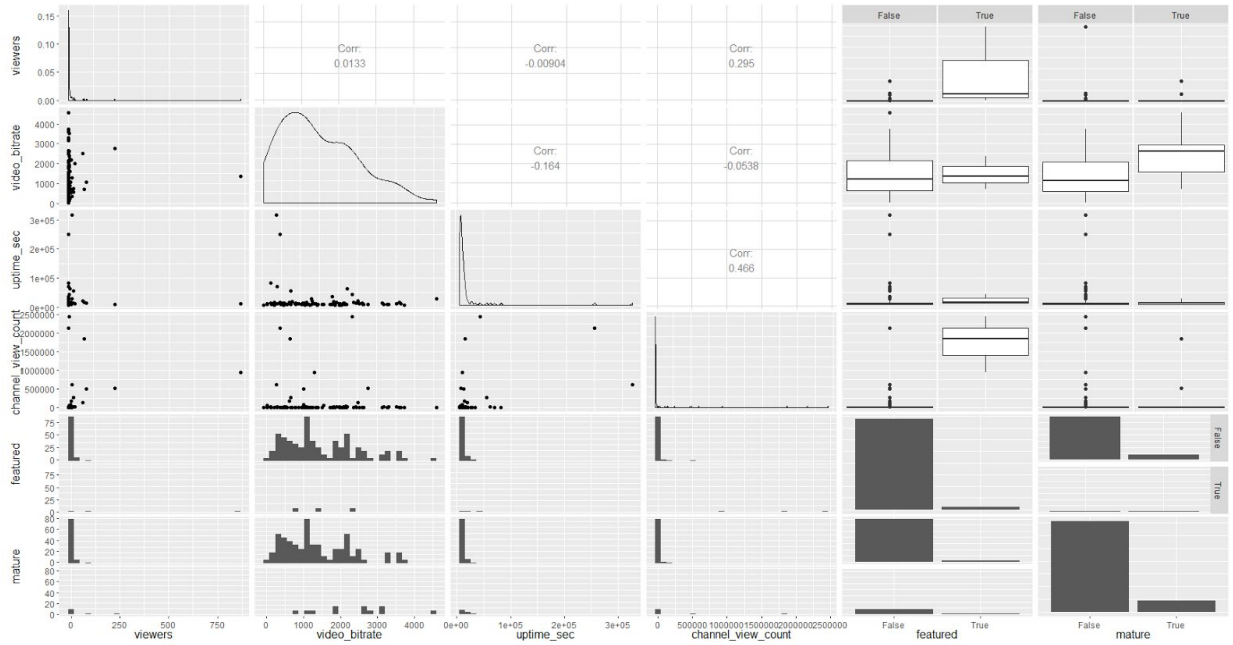


Fig 1: GGPairs Plot of all relevant variables

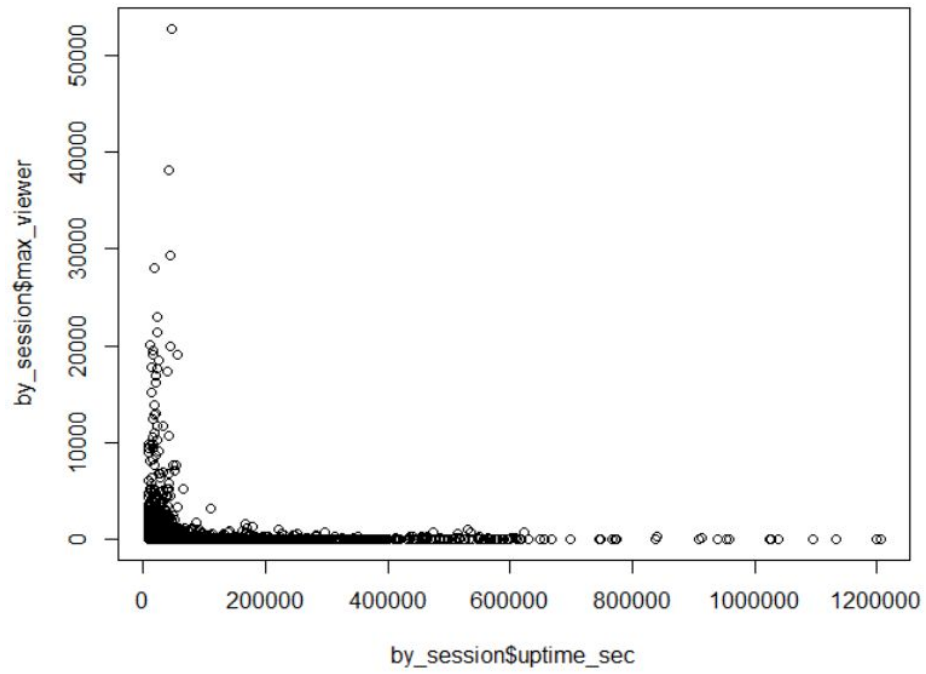


Fig 2: scatterplot of a covariate vs. response variable

Modeling approaches

For regression modeling, we tried 3 approaches and recorded the 5-fold CV error for each method:

- Linear Regression

  Coefficients:

(Intercept) video_bitrate uptime uptime_sec language_session channel_view_count broadcaster
1.125e+05 1.543e-03 -7.093e+00 1.411e-04 -9.864e-02 5.913e-05 -8.161e-02
featured channel_subscription mature US
4.427e+01 6.848e+02 -4.305e-01 -4.060e+00

CV error = 157.44

- Lasso Regression
  After standardizing the covariates and the response variable, we applied Lasso Regression, with varying penalty threshold, to see if we could get an improvement over OLS on the CV error

Lambda	0.1	0.5	1
CV Error	158.89	160.63	183.06

- Random Forest

We also tried a discrete classification method, random forest, to model the number of peak viewers in response to the 11 attributes of the streaming session. We experimented with different number of buckets, n_tree = 5, 10, 20. This method turned out very inaccurate for all the thresholds. We postulate that because of the spread of the data, where most sessions receive double-digit viewers whereas some reach tens of thousands of viewers, dividing the response variable into buckets fails to distinguish the difference between most of the data points.

Best Model Evaluation

Based on the 5-fold CV error, we chose Linear Regression to be our best model because it yielded the lowest CV error. Adding penalty thresholds in Lasso did not seem to improve accuracy of prediction, most likely because we have already pre-selected and removed irrelevant and highly dependent variables. The discrete method we tried, Random Forest, led to inaccurate results.

Since the model was trained and tested on the same set of data (data over a week) and the test set we plan on using will be data from another week, we expect the generalization/test error to increase because of the differences between train and test data, as they are taken from slightly different time period. However, the train set is the most faithful representation we could find so we don't expect any unreasonable spike in generalization error.

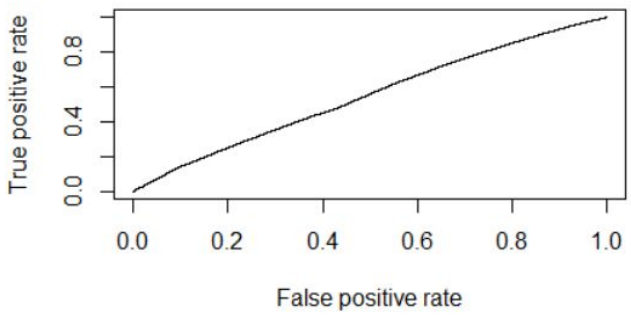
Classification Task

Baseline

The simplest baseline is always returning False for primetime, and it would actually achieve > 70% accuracy (average 0-1 loss) in the training dataset, since there are so much more sessions not in primetime than the sessions within primetime.

We created another baseline of including all covariates that make sense, with no interaction or higher order terms.

This baseline created a following ROC curve using our training dataset:



As you can see, the curve is very close to the line y=x and the model is not able to really distinguish the two possibilities (primetime vs otherwise).

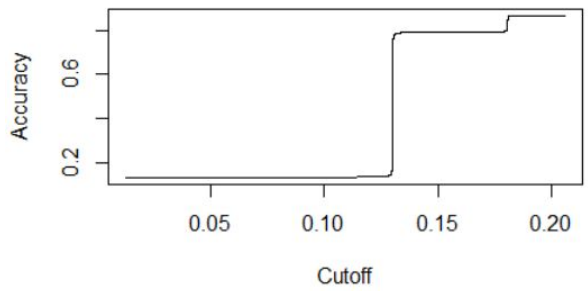
As expected, the AUC was very close to 0.5. Our calculated AUC for this baseline was 0.5469.

Modeling Approach & Evaluation

We did not intuitively think that k-nearest-neighbors algorithm applies well to our situation and some of our important covariates were numerical, which excluded Naive Bayes (at least the form we learned so far). Therefore, we built several models, all using logistic regression but with different interaction terms and transformations. We present one of the models below. Difference from baseline: Interaction terms with max_viewer:channel_view_count, mature:featured, and max_viewer:uptime_sec. This modification seemed to make sense since these variables most likely cooperate to have some correlation on whether the session was streamed at primetime or not.

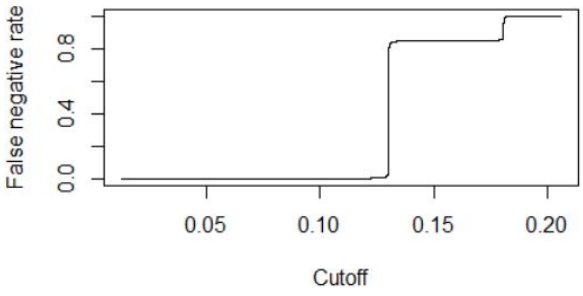
However, most interaction terms/transformations we performed did not really seem to affect the performance too well, at least what we could glean from the ROC curve.

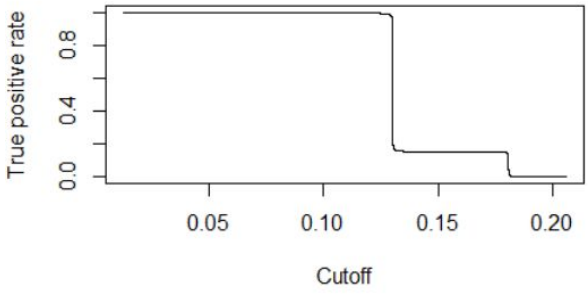
We plotted the accuracy for different cutoff values (the minimum output of regression needed to classify as a “true” value, or a 1):

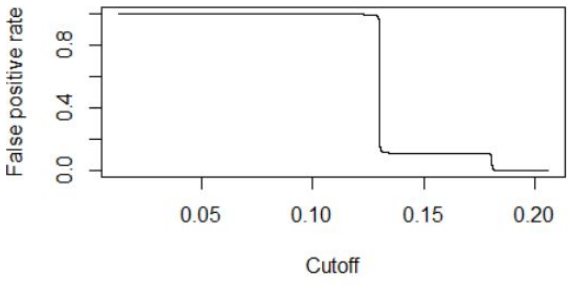


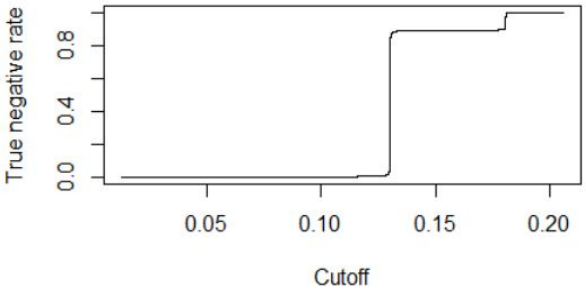
Because the data is heavily skewed to non-primetime sessions, accuracy does not really reflect a good measure of performance. If the primetime classification is used anywhere, it will most likely be used to figure out factors a person might consider when deciding when to hold a Twitch TV streaming session.

With that in mind, it is important to be able to interpret the coefficients to our model. In addition, goal should be to reduce false negative rate as much as possible, so that no shows that are good for primetime showing are ignored, even if some shows that might not fit the bill make it to the list. Reducing false negatives while increasing true positives is important even if it costs us true negatives and false positives. Therefore, I generated confusion matrix values vs cutoff graphs:

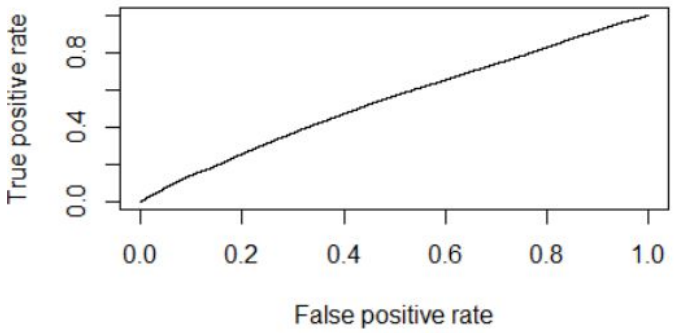








This suggested that this model in general had no good separation between primetime and non-primetime shows. I then checked the ROC curve on the training data and the corresponding AUC:



As expected, the ROC curve did not look like it was representing a good classifier, and the AUC value was 0.544. Regardless, a good cutoff value for this model certainly lies between 0.12 and 0.15, where it has some hope of discriminating true from false primetime tags.

Approximation of Prediction Error

The best model is the model presented above, which was not much better than the baseline in many ways. Other modeling approaches gave similar profiles for AUC/ROC curve and optimal cutoff points.

We were not yet completely sure what the best measure of success for this classification problem, so we report the accuracy as a validation measure here. We approximate our general accuracy to be around 0.79 in the test set. This number was calculated by dividing the training data into a validation and a training set, and a K-fold approach would improve this number. This time we were not able to perform a K-fold validation approach, and we will have to think of better performance metric/methods.

```
##### Regression Code #####

library(GGally)
library(chron)
library(cvTools)
load('C:/Users/Joseph/Documents/R/cleanData')

#removing NA and empty entries#
by_session = by_session[-which(by_session$uptime == ""), ]
by_session = by_session[-which(is.na(by_session$max_viewer)), ]
by_session$US = by_session$geo == 'US'
by_session = subset(by_session, select = -c(geo))

cleaned_timestamp = c()
for (i in 1:nrow(by_session)) {
  timestamp = toString(by_session$uptime[i])
  ts = unlist(strsplit(timestamp, "\\s+"))
  cleaned_timestamp[i] = chron(paste(ts[2], ts[3], ts[5]), ts[4], format = c('mon d y', 'h:m:s'))
}

by_session$uptime = cleaned_timestamp


# sample mean predictions #
sample_mean = mean(by_session$max_viewer)
RMSE_sm = sqrt(mean((by_session$max_viewer - sample_mean)^2))

# observe correlation of covariates#
cor(data.matrix(by_session))

# linear regression #
by_session.reg = subset(by_session, select = -c(session_id, channel_id, language_channel,
timezone, producer))
by_session.reg = as.data.frame(data.matrix(by_session.reg))
fm = lm(max_viewer ~ ., data = by_session.reg)
cv.lm = cvFit(fm, data=by_session.reg, y=by_session.reg$max_viewer, K=5)


##### Classification Code ######
library(chron)
library(ROCR)
library(boot)
# Clean Data ----
```

```
by_session = by_session[-which(by_session$uptime == ""), ]

cleaned_timestamp = rep(chron(), nrow(by_session))
primetime = vector(length=nrow(by_session))
for (i in 1:nrow(by_session)) {by
  timestamp = toString(by_session$uptime[i])
  ts = unlist(strsplit(timestamp, "\\s+"))
  temp = chron(paste(ts[2], ts[3], ts[5]), ts[4], format = c('mon d y', 'h:m:s'))
  time = strsplit(ts[4], ':')
  if (as.numeric(time[[1]][1]) > 18 && as.numeric(time[[1]][1]) < 22) {
    primetime[i] = 1;
  }
  cleaned_timestamp[i] = temp
}

by_session$uptime = cleaned_timestamp
by_session$primetime = primetime
classification_data = by_session
classification_data$mature = as.logical(classification_data$mature)*1
classification_data$producer = as.logical(classification_data$producer)*1
classification_data$channel_subscription =
as.logical(classification_data$channel_subscription)*1
classification_data$featured = as.logical(classification_data$featured)*1
classification_data$viewed = (classification_data$max_viewer > 3)*1


# Primetime Classification Task -----

# Baseline

fm_baseline = glm(primetime ~ max_viewer + video_bitrate + uptime_sec +
channel_view_count + featured + mature + channel_subscription,
family='binomial',data=classification_data)

prob = predict(fm_baseline1, classification_data, type='response')
pred = prediction(prob, classification_data$primetime)

fnr.perf = performance(pred, measure = "fnr")
tpr.perf = performance(pred, measure = "tpr")
tnr.perf = performance(pred, measure = "tnr")
fpr.perf = performance(pred, measure = "fpr")
plot(fnr.perf)
plot(tpr.perf)
```

```
plot(fpr.perf)
plot(fpr.perf)

# Get ROC curve and AUC
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
auc <- performance(pred, measure = "auc")
auc <- auc@y.values[[1]]
auc

# Model 1

fm_model_1 = glm(primetime ~ max_viewer + uptime_sec + channel_view_count + featured +
mature + max_viewer:channel_view_count + mature:featured + max_viewer:uptime_sec,
family='binomial',data=classification_data)

prob = predict(fm_model_1, classification_data, type='response')
pred = prediction(prob, classification_data$primetime)

acc.perf = performance(pred, measure = "acc")
plot(acc.perf)

fnr.perf = performance(pred, measure = "fnr")
tpr.perf = performance(pred, measure = "tpr")
tnr.perf = performance(pred, measure = "tnr")
fpr.perf = performance(pred, measure = "fpr")
plot(fnr.perf)
plot(tpr.perf)
plot(fpr.perf)
plot(tnr.perf)

# Get ROC curve and AUC
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
auc <- performance(pred, measure = "auc")
auc <- auc@y.values[[1]]
auc

# Divide into train/validate
val_rows = sample(nrow(classification_data), 50000)
val_set = classification_data[val_rows, ]
train_set = classification_data[-val_rows, ]
```

```
fm_model_1 = glm(primetime ~ max_viewer + uptime_sec + channel_view_count + featured +
mature + max_viewer:channel_view_count + mature:featured + max_viewer:uptime_sec,
family='binomial',data=train_set)
prob = predict(fm_model_1, val_set, type='response')
preds = (prob > 0.15)*1
correct = 0
for (i in 1:nrow(val_set)){
  if (!is.na(preds[i]) && !is.na(val_set$primetime[i])){
    if (preds[i] == val_set$primetime[i]) {
      correct = correct + 1
    }
  }
}
accuracy = correct/50000
accuracy
```

Prediction on the test set

Regression prediction on test set

Best Model: OLS	Prediction Error (RMSE)
Train Set	157.44
Test Set	190.62

We used RMSE (Root Mean Squared Error) to measure the prediction error of the regression model on both train and test sets. We used cross validation method to estimate the true error from the train set. Notice that the prediction error is still quite a bit higher than the prediction error on the train set, even with the cross validation method. However, the prediction error is lower than our baseline error, showing that our regression model indeed improves prediction accuracy.

Classification prediction on test set

Best Model: Logistic Regression	Accuracy
Approximation of Test Accuracy	0.79
Calculated Test Accuracy	0.80

The true test error when our fitted logistic regression model was applied to the brand-new test set was very close to our approximation of test accuracy. This gives us confidence that what we say about our model just using the training data generalizes to the world outside of the test data relatively well.

This test accuracy may not hold if a significant amount of time passes between the time when the training data set was collected and the test data set was collected. Here, we used datasets

that are a week apart, but if the time difference grows to weeks and months, we may need to refit the model to make the training data better look like the incoming, new data. We discuss this further in the discussion.

Inference

We decided to implement the following inference methods on our regression task (predicting the continuous variable - number of peak viewers during a streaming session). See our linear regression model below:

	Coefficient	P-Value
(Intercept)	1.486052e+00	0.40193
Video_bitrate	1.564666e-03	0.00115
Uptime	-1.251898e+00	0.53251
Uptime_sec	2.203679e-04	4.43e-15
Channel_view_count	5.915931e-05	< 2e-16
Featured	4.388593e+01	2.32e-14
Channel_subscription	6.846699e+02	< 2e-16
Mature	-3.804961e-01	0.81848
US	-2.598272e+00	0.03061
Eng_lang	2.256840e+00	0.15083

Part A) Using a p-value threshold of 0.05, we find that Video_bitrate, Uptime_sec, Channel_view_count, Featured, Channel_subscription, and the label US to be statistically significant coefficients. Many of these coefficients, especially channel_view_count, featured, and channel_subscription, are readily interpretable. Here, significance for a coefficient means that if this coefficient was actually 0 (the covariate does not affect viewer_count in any way) there is a p-value chance of seeing the data as it happened. We can believe the results for the covariates mentioned above, but video_bitrate, uptime_sec, and the US label seem to be rather spurious, and are not as readily interpretable.

Part B) After fitting our model on the test data, we looked at the coefficients and their corresponding P values. Although the coefficients are slightly different, the same significant coefficients held (uptime, featured, channel_subscription, mature, US, eng_lang), showing that indeed these attributes have a strong correlation to the number of viewers who tune in during a session.

Part C) After bootstrapping 5000 times, we used the standard error from bootstrap distribution to estimate a 95% Confidence Interval for each coefficient:

	95% Confidence Interval	
(Intercept)	1.486052e+00	± 4.199989e+00
Video_bitrate	1.564666e-03	± 7.430702e-04
Uptime	-1.251898e+00	± 3.133581e+00
Uptime_sec	2.203679e-04	± 7.947176e-05
Channel_view_count	5.915931e-05	± 2.86392e-05
Featured	4.388593e+01	± 7.704256e+01

Channel_subscription	6.846699e+02	± 1.26249e+02
Mature	-3.804961e-01	± 3.745552e+00
US	-2.598272e+00	± 2.334811e+00
Eng_lang	2.256840e+00	± 4.119952e+00

Side by side comparison of R generated Std Error and Bootstrap Std Error:

	R Std Error	Bootstrap Std Error
(Intercept)	1.773e+00	2.142851e+00
Video_bitrate	4.813e-04	3.791174e-04
Uptime	2.006e+00	1.598766e+00
Uptime_sec	2.810e-05	4.054682e-05
Channel_view_count	5.408e-07	1.461184e-05
Featured	5.750e+00	3.930743e+01
Channel_subscription	8.110e+00	6.441278e+01
Mature	1.658e+00	1.910996e+00
US	1.202e+00	1.19123e+00
Eng_lang	1.571e+00	2.102016e+00

We notice slight differences in the standard error of every coefficient but the R generated std error and Bootstrap std error are generally in the same neighborhood. The difference result because R employs a different methodology to estimate the std error than the Boostrap method. R relies on the single sample input into the linear model to estimate the std error of the coefficients generated. Bootstrap is a way to artificially generate lots of samples from the original sample and uses the standard deviation of the distribution to estimate the std error.

Part D) Because we have either removed or converted irrelevant variables and categorical variables, we reduced the number of covariates to 9. Adding back all the covariates that can be input into OLS is essentially creating noise in the algorithm. The new coefficients with all the covariates fitted show minimal changes in the original coefficients:

	Coefficient (9 Cov)	Coefficient (14 Cov)
(Intercept)	1.486052e+00	1.424352e+00
Video_bitrate	1.564666e-03	1.335672e-03
Uptime	-1.251898e+00	-1.244891e+00
Uptime_sec	2.203679e-04	2.203679e-04
Channel_view_count	5.915931e-05	5.912264e-05
Featured	4.388593e+01	4.438519e+01
Channel_subscription	6.846699e+02	6.832492e+02
Mature	-3.804961e-01	-3.603943e-01
US	-2.598272e+00	-2.342986e+00
Eng_lang	2.256840e+00	2.003835e+00
Session_id		2.536339e-02
Channel_id		8.795637e-04
Video_width		5.758951e-04
Embed_count		5.221779e-05
Site_count		2.2359678e-04

Part E) We found multiple coefficients as significant, and given that we started out with 13+ coefficients, it would not be unlikely that at least one of the coefficients we found significant is actually a false positive. The Bonferroni correction would make our effective pvalue around 0.006, in which case the video_bitrate and the US label would not make it as significant coefficients. This actually seems reasonable, because we were not completely convinced that these variables display true significance. Our model building involved a subjective step where we incorporated variables that we thought were significant, but this step did not use our data; in other words, our model selection procedure did not involve going back to the data multiple times. Because of this, we are fairly confident that we are not subject to post selection inference, aside from the possibility that some actually significant covariates were excluded based on our false intuition

Part F) In our context, the "featured" variable indicated whether Twitch TV had shown the session on the first landing page of this website (Figure 1). From this exogenous data and our significant coefficient and p value, we are inclined to say that there is a causal relationship between being featured and the viewer count of a session. Being on the prime real estate of the website's landing page would reasonably lead to more publicity and promotion, thus more viewers. There may be other variables in the data that lead to a session being featured. However, we have no idea what the selection criteria for a session to be featured on Twitch TV's home page; therefore the featured variable is not representative of a random assignment. Because we may have a sampling bias that we cannot confirm or deny, just using our dataset, we cannot give quantitative evidence for causality even between the featured variable and viewer count.

Additionally, we are inclined to assert a causal relationship between whether a channel takes subscriptions (the "channel_subscription" variable) and the viewer count of its session. On Twitch, streamers have the option to make their channel into a subscription channel, where the audience can subscribe and make donations to. Subscribers of a channel can view the streams when logging into their account and receive notifications of new streams (Figure 2). This gives a huge advantage in terms of reaching fans and promoting a channel's content. Since this subscription option is an independent decision that a streamer makes when creating a channel, we don't believe there are confounding variables within the data. An argument can be possibly made that streamers who take subscribers produce more consistent and better content. However, these are variables that are outside of the data we used.

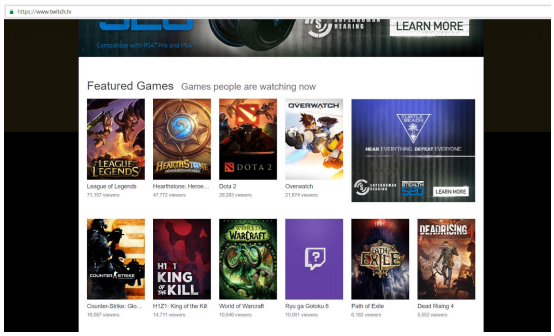


Figure 1

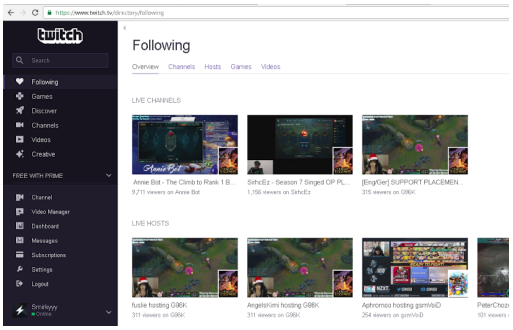


Figure 2

Discussion

Our dataset comes from an online streaming TV source. As the administrators of the website, there could be many questions that they would like to answer about the user behaviors on the platform. These questions would be answered by both prediction and inference. For example, they may simply want to predict the viewer count in various sessions held at specific times, to estimate what kind of backend server bandwidth they need; depending on how their backend is implemented, it could lead to business decisions involving hardware rental/purchase. Our regression model for prediction could be a starting step in answering this question. In addition, the site admins may want to find out what different factors make a significant role in contributing to the viewer count, so they can focus on that factor if they are able to. One example might be adjusting their criteria for promoting “hot” or “featured” sessions, so that the ones that actually do get featured can get maximum benefit in terms of viewer count.

Our model would hold up within intervals of time with similar properties. For example, a model fitted for the holiday season would probably hold up for the next few months when people maintain their same habits before the holidays are over; the model would most likely have to be refitted every time there is a significant seasonal change that prompts people to change their behaviors. In addition to seasons, when there are important gaming events (League of Legends tournaments, etc.) our model may have to be refitted to reflect that people who usually do not use the platform are now active.

We would definitely notify anyone who uses our model of how we organized the raw data into the data frame that we ultimately ended up performing analysis with. We organized the raw event log data by sessions, so that each session was an “observation” in the design matrix. We also defined the viewer count of each session as the maximum viewer count observed at any given point of the session - if one is interested in the behavior of the viewer count within a session, our cleaned dataset would not contain information about that. We may also have issues with multiple hypothesis testing, since we do have around 15 covariates - it would not be anomalous if one or two of these covariates to show up as significant just because of pure chance, not because of their actual underlying link to the viewer count. In addition, we manually chose the covariates in the beginning based on our intuition that this variable would

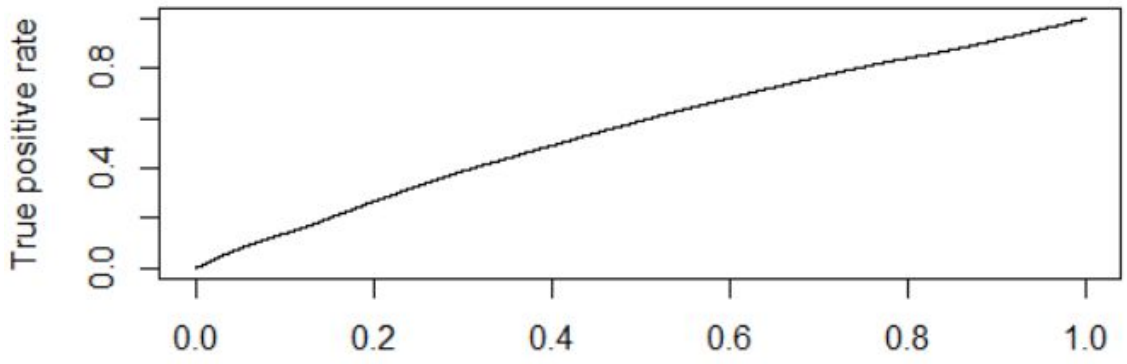
or would not matter - by performing this process, we might be vulnerable to post-selection inference. A more robust analysis would try to account for all these potential roadblocks.

We did not collect the data so we cannot really affect the data collection process, but if we could, we would collect more covariates that may be helpful in building practical models. For example, Overall traffic through the site at a given time would be a reasonable covariate that may help with our analysis, so that we can normalize the “relative popularity,” or what percentage of online viewers were at the site because of this specific session. In addition, we would make data collection more uniform over time. We noticed that information is not collected over uniform intervals of time, which made the data cleaning process a bit confusing.

If we started the project over again, we may change the data cleaning process so that we can capture information about the viewer count within a session over time. The way we did it this time loses all temporal resolution within the session, and it may interesting to see what kind of features makes sessions lose viewers quickly (even if a session started with a high viewer count). We believe we still would have performed similar viewer count type of analysis.

Appendix:

ROC curve for classification on test set (what would have happened if I changed my threshold):



#removing NA and empty entries#
by_session = by_session[-which(by_session$uptime == ""),]

```
by_session = by_session[-which(is.na(by_session$max_viewer)), ]

cleaned_timestamp = c()
for (i in 1:nrow(by_session)) {
  timestamp = toString(by_session$uptime[i])
  ts = unlist(strsplit(timestamp, "\\s+"))
  cleaned_timestamp[i] = chron(paste(ts[2], ts[3], ts[5]), ts[4], format = c('mon d y', 'h:m:s'))
}

by_session$uptime = cleaned_timestamp

by_session$US = by_session$geo == 'US'
by_session = subset(by_session, select = -c(geo))

by_session$eng_lang = by_session$language_session == 'en'
by_session$channel_subscription = by_session$channel_subscription == 'True'
by_session$featured = by_session$featured == 'True'
by_session$mature = by_session$mature == 'True'
by_session = subset(by_session, select = -c(language_session, broadcaster))

by_session.reg = subset(by_session, select = -c(session_id, channel_id, language_channel,
timezone, producer))
by_session.reg = as.data.frame(data.matrix(by_session.reg))
by_session.reg$uptime = by_session.reg$uptime %% 1
fm = lm(max_viewer ~ ., data = by_session.reg)
fm$coefficients

# Bootstrap to generate CI #
# bootstrap 5000 times #
coef_bootstrap = matrix( rep( 0, 10*5000), nrow = 5000, ncol = 10)
for (i in 1:5000) {
  bs_sample = by_session.reg[sample(nrow(by_session.reg), size = nrow(by_session.reg),
replace = TRUE),]
  fm = lm(max_viewer ~ ., data = bs_sample)
  coef_bootstrap[i,] = fm$coefficients
}

library(chron)
library(ROCR)

#by_session$uptime = cleaned_timestamp
classification_data = by_session
classification_data$mature = as.logical(classification_data$mature)*1
```

```
classification_data$producer = as.logical(classification_data$producer)*1
classification_data$channel_subscription =
as.logical(classification_data$channel_subscription)*1
classification_data$featured = as.logical(classification_data$featured)*1
classification_data$viewed = (classification_data$max_viewer > 3)*1

# Fit best model 1
fm_model_1 = glm(primetime ~ max_viewer + uptime_sec + channel_view_count + featured +
mature + max_viewer:channel_view_count + mature:featured + max_viewer:uptime_sec,
family='binomial',data=classification_data)

# Test on test data set

# Process the test data set a bit
primetime_test = vector(length=nrow(by_session_test))
for (i in 1:nrow(by_session_test)) {
  timestamp = by_session_test$uptime[i]
  ts = unlist(strsplit(timestamp, "\\s+"))
  time = strsplit(ts[4], ':')
  if (as.numeric(time[[1]][1]) > 18 && as.numeric(time[[1]][1]) < 22) {
    primetime_test[i] = 1;
  }
}

by_session_test$primetime = primetime_test
test_data = by_session_test
test_data$mature = as.logical(test_data$mature)*1
test_data$producer = as.logical(test_data$producer)*1
test_data$channel_subscription = as.logical(test_data$channel_subscription)*1
test_data$featured = as.logical(test_data$featured)*1
test_data$viewed = (test_data$max_viewer > 3)*1


# Predict on test data
prob = predict(fm_model_1, test_data, type='response')
preds = (prob > 0.15)*1
correct = 0
total = 0
for (i in 1:nrow(by_session)){
  if (!is.na(preds[i]) && !is.na(test_data$primetime[i])){
    total = total + 1
    if (preds[i] == test_data$primetime[i]) {
      correct = correct + 1
```

```
    }
  }
}
accuracy = correct/total
# Accuracy:
print(accuracy)

# Get ROC curve and AUC
pred = prediction(prob, test_data$primetime)
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
auc <- performance(pred, measure = "auc")
auc <- auc@y.values[[1]]
auc
```